From disorder to insight: harnessing entropy

Jos Prinsen, Caterina Ceccato, Ethel Pruss, Anita Vrins

2023-05-31

Basic idea of Entropy

In information theory, entropy is a measure of uncertainty or randomness in a random variable or a probability distribution. It quantifies the average amount of information or surprise that is needed to describe an event drawn from that distribution (Shannon, C. E., 1948). In our case, we are applying these theories to time series data.

To start off, we will be using Shannon entropy to provide insights into the complexity of the EEG signal. Shannon entropy is a measure of the average amount of information or uncertainty present in data (Saraiva, P, 2023; Shannon, C. E., 1948). It quantifies the degree of randomness or unpredictability in a probability distribution. One can imagine a signal that is constantly the same and almost never changes it's value, has a low Entropy. Contrary, a signal that is constantly changing, and has different values every time has high entropy. A signal has maximized entropy, when each state is unique, and the probability of observing each state is equal.

Shannon Entropy is calculated as follows:

1. **Start with a probability distribution:** Shannon entropy measures the uncertainty or randomness in a probability distribution. Begin by identifying the probabilities of each possible outcome in the distribution. The sum of these probabilities should be 1.

2. **Calculate the information content:** For each outcome, calculate its information content using this formula: Information content = -log2(p)

   "p" represents the probability of that outcome. The logarithm base 2 is commonly used to measure information in bits.

3. **Multiply information content by probabilities:** Multiply each outcome's information content by its probability.

4. **Sum up the results:** Add up the products obtained in step 3 for all outcomes.

5. **Calculate the Shannon entropy:** Finally, calculate the Shannon entropy by taking the sum from the previous step and making it negative

   Shannon entropy = - Σ (p * log2(p))

   Here, Σ denotes the sum, "p" represents the probability of each outcome, and log2(p) represents the logarithm base 2 of the probability.

Here we calculate the Shannon Entropy for the Fz channel in 1 epoch for the first participant.

```r
# Create a vector of channel names
channellist <- c("Fz", "C3", "Cz", "C4",
    "Pz", "PO7", "Oz", "PO8")
```

```r
# Access the first element of the first
# element of
# RawEEGComplexSystemsRReadyAdaptiveR
# list corresponding to the 'Fz'
# channel
AdaptiveSignal <- RawEEGComplexSystemsRReadyAdaptiveR[[1]][[1]][["Fz"]]

# Compute the relative frequency of
# each unique value in AdaptiveSignal
# and create a table
p1AOriginal <- table(AdaptiveSignal)/length(AdaptiveSignal)

# Compute the entropy of p1AOriginal
# using the formula -sum(p * log2(p))
e1AOriginal <- -sum(p1AOriginal * log2(p1AOriginal))

# Print the probability of the first 5
# samples and the total entropy value
cat("Probability of first 10 samples: ",
    head(p1AOriginal, 10), "\n Total entropy value: ",
    e1AOriginal)
```

```
## Probability of first 10 samples:  0.0003904725 0.0003904725 0.0003904725 0.0003904725 0.0003904725 0
##  Total entropy value:  11.32249
```

As you can see the probabilities of each sample are exactly the same. This means that the entropy is maximized for this length signal. Since our EEG data is continuous data measured with several decimals, the chance of the EEG signal being exactly the same is extremely small. Therefore each state has the same probability of occurring, which means that the signal is as uncertain as it can be.

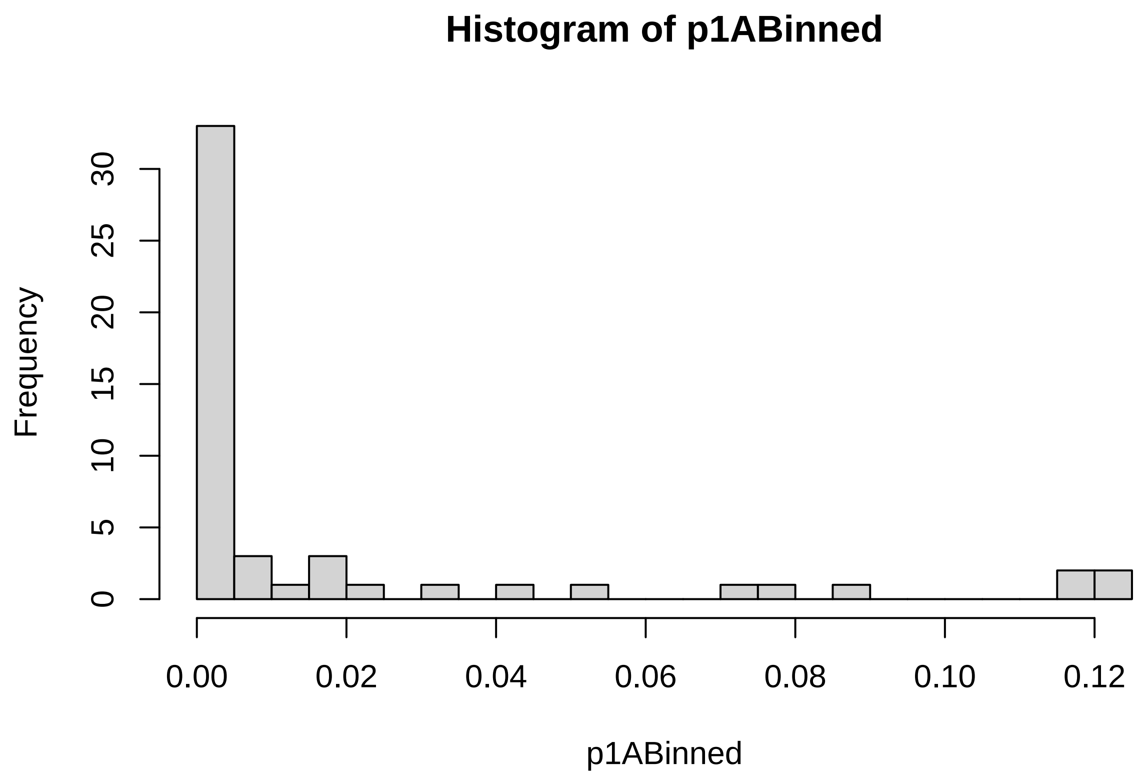
Binning, also known as discretization, is a process of grouping continuous data into discrete intervals or bins (Navas-Palencia, G., 2020). Values that are close together that fall within the same Interquartile range (IQR) are grouped together. This method reduces the complexity of the data, which allows values that are extremely close together to be grouped together, and be considered as “the same variable”.

```r
# Create a vector of channel names
channellist <- c("Fz", "C3", "Cz", "C4",
    "Pz", "PO7", "Oz", "PO8")

# Define the number of bins for binning
num_bins <- 64

# Access the first element of the first
# element of
# RawEEGComplexSystemsRReadyAdaptiveR
# list corresponding to the 'Fz'
# channel
AdaptiveSignal <- RawEEGComplexSystemsRReadyAdaptiveR[[1]][[1]][["Fz"]]

# Compute the breakpoints for binning
# based on the minimum and maximum
# values of AdaptiveSignal
breaks <- seq(min(AdaptiveSignal), max(AdaptiveSignal),
    length.out = num_bins + 1)

# Bin the values of AdaptiveSignal into
# the defined number of bins
binnedSignal <- cut(AdaptiveSignal, breaks = breaks,
    include.lowest = TRUE, labels = FALSE)

# Compute the relative frequency of
# each unique value in binnedSignal and
# create a table
p1ABinned <- table(binnedSignal)/length(binnedSignal)

# Compute the entropy of p1ABinned
# using the formula -sum(p * log2(p))
e1ABinned <- -sum(p1ABinned * log2(p1ABinned))

# Print the probability per bin and the
# total entropy value

# Print probability per bin and total
# entropy value
cat(" Probability per bin: ", p1ABinned,
    "\n Total entropy value: ", e1ABinned)
```

```
##  Probability per bin:  0.001171417 0.00156189 0.00468567 0.005076142 0.003904725 0.00312378 0.002733
##  Total entropy value:  3.954579
```

As you can see, the entropy is way lower than that of the original signal. We can also see that there are some bins that occur very rarely. When the probabilities are very small, it means that the corresponding outcomes have a very low likelihood of occurring, and the information content associated with those outcomes is quite high. So, when the probabilities are very small, the information content becomes high, indicating that these outcomes carry a significant amount of surprise or unexpectedness. It suggests that those rare events, although unlikely to occur, would provide more information if they did happen.

```r
hist(p1ABinned, breaks = length(unique(p1ABinned)))
```



This histogram shows that there are many bins that have a low likelihood but provide quite some information. It would be interesting to investigate a longer EEG signal. We saw in the Multifractal chapter that EEG spectral data is scale-free, meaning it has a 1/f distribution. It could be the case that entropy describing a long EEG signal will also follow a similar pattern. To investigate this, the number of bins would have to be increased, and a way longer EEG signal would have to be analyzed. An analysis like this could also show the opposite trend, where we can see that the number of low likelihood- high information will diminish, as in the current analysis it is possible that they are created due to residual noise.

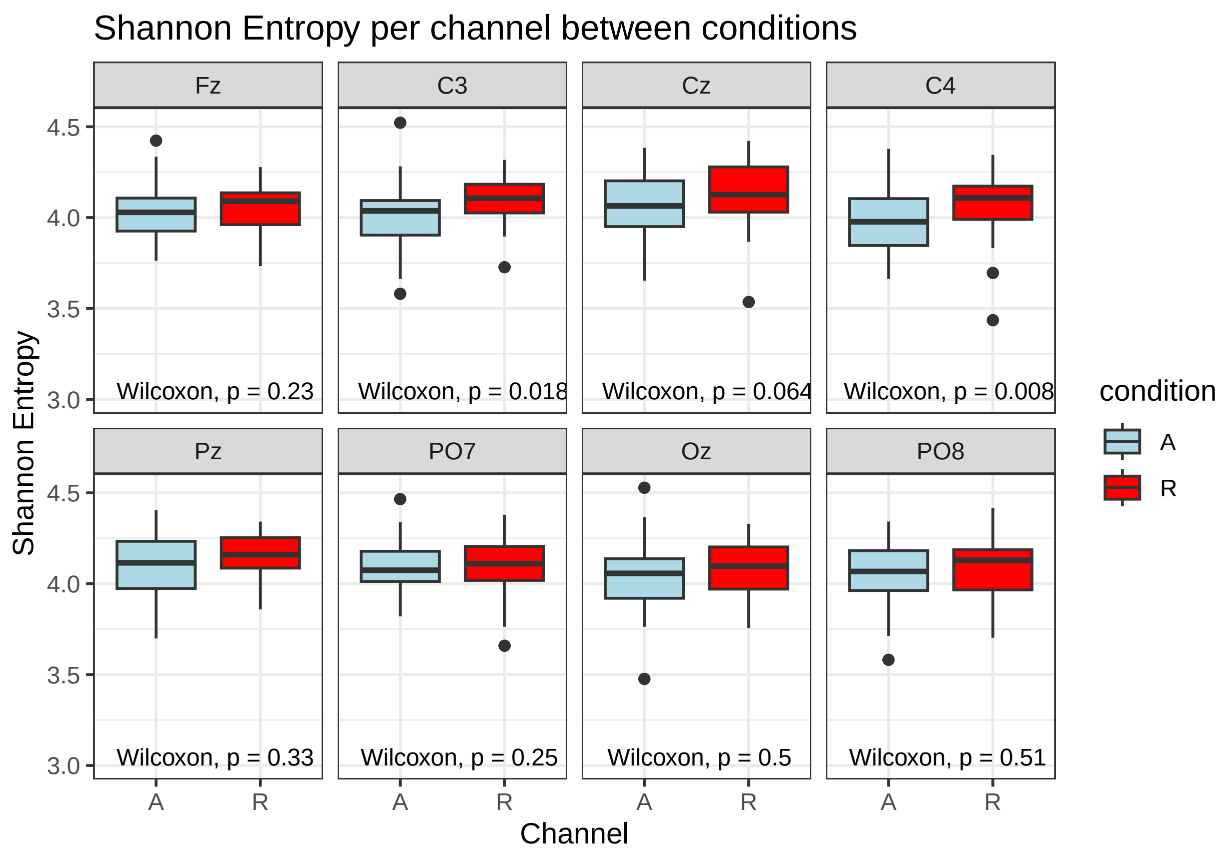
Shannon Entropy using Bin clustering (30) over all participants

```r
# Define the list of channels
channellist <- c("Fz", "C3", "Cz", "C4",
    "Pz", "PO7", "Oz", "PO8")

# Number of bins for binning
num_bins <- 30

# Lists to store Shannon entropy values
ShannonAdaptiveChannels <- list()
ShannonAdaptiveEvents <- list()
ShannonAdaptiveParticipants <- list()

ShannonRandomChannels <- list()
ShannonRandomEvents <- list()
```

```r
ShannonRandomParticipants <- list()

# Calculate Shannon entropy for
# adaptive participants
for (i in 1:38) {
    ShannonAdaptiveEvents <- vector(mode = "list",
        length = length(RawEEGComplexSystemsRReadyAdaptiveR[[i]]))
    for (A in 1:length(RawEEGComplexSystemsRReadyAdaptiveR[[i]])) {
        for (channel in channellist) {
            AdaptiveSignal <- RawEEGComplexSystemsRReadyAdaptiveR[[i]][[A]][[channel]]

            # Binning the signal
            breaks <- seq(min(AdaptiveSignal),
                max(AdaptiveSignal), length.out = num_bins +
                  1)
            binnedSignal <- cut(AdaptiveSignal,
                breaks = breaks, include.lowest = TRUE,
                labels = FALSE)

            p1A <- table(binnedSignal)/length(binnedSignal)
            e1A <- -sum(p1A * log2(p1A))

            ShannonAdaptiveChannels[[channel]] <- e1A
        }

        ShannonAdaptiveEvents[[A]] <- ShannonAdaptiveChannels
    }
    # Calculate Shannon entropy for
    # random participants
    if (i != 38) {
        ShannonRandomEvents <- vector(mode = "list",
            length = length(RawEEGComplexSystemsRReadyRandomR[[i]]))
        for (R in 1:length(RawEEGComplexSystemsRReadyRandomR[[i]])) {
            for (channel in channellist) {
                RandomSignal <- RawEEGComplexSystemsRReadyRandomR[[i]][[R]][[channel]]
                breaks <- seq(min(RandomSignal),
                  max(RandomSignal), length.out = num_bins +
                    1)
                binnedSignalR <- cut(RandomSignal,
                  breaks = breaks, include.lowest = TRUE,
                  labels = FALSE)
                p1R <- table(binnedSignalR)/length(binnedSignalR)

                e1R <- -sum(p1R * log2(p1R))

                ShannonRandomChannels[[channel]] <- e1R
            }
            ShannonRandomEvents[[R]] <- ShannonRandomChannels
        }

        ShannonRandomParticipants[[i]] <- ShannonRandomEvents
    }
    ShannonAdaptiveParticipants[[i]] <- ShannonAdaptiveEvents
}

# Unlist the Shannon entropy values
continuousListA <- unlist(ShannonAdaptiveParticipants,
    recursive = FALSE)
continuousListR <- unlist(ShannonRandomParticipants,
    recursive = FALSE)

resultListA <- list()
resultListR <- list()
for (i in seq_along(continuousListA)) {
    resultListA[[i]] <- unlist(continuousListA[[i]])
}
for (i in seq_along(continuousListR)) {
    resultListR[[i]] <- unlist(continuousListR[[i]])
}

# Extract Shannon entropy values for
# each channel
Fz_listA <- unlist(lapply(resultListA, function(a) a["Fz"]),
    use.names = FALSE)
C3_listA <- unlist(lapply(resultListA, function(a) a["C3"]),
    use.names = FALSE)
Cz_listA <- unlist(lapply(resultListA, function(a) a["Cz"]),
    use.names = FALSE)
C4_listA <- unlist(lapply(resultListA, function(a) a["C4"]),
    use.names = FALSE)
Pz_listA <- unlist(lapply(resultListA, function(a) a["Pz"]),
    use.names = FALSE)
PO7_listA <- unlist(lapply(resultListA, function(a) a["PO7"]),
    use.names = FALSE)
Oz_listA <- unlist(lapply(resultListA, function(a) a["Oz"]),
    use.names = FALSE)
PO8_listA <- unlist(lapply(resultListA, function(a) a["PO8"]),
    use.names = FALSE)

Fz_listR <- unlist(lapply(resultListR, function(a) a["Fz"]),
    use.names = FALSE)
C3_listR <- unlist(lapply(resultListR, function(a) a["C3"]),
    use.names = FALSE)
Cz_listR <- unlist(lapply(resultListR, function(a) a["Cz"]),
    use.names = FALSE)
C4_listR <- unlist(lapply(resultListR, function(a) a["C4"]),
    use.names = FALSE)
Pz_listR <- unlist(lapply(resultListR, function(a) a["Pz"]),
    use.names = FALSE)
PO7_listR <- unlist(lapply(resultListR, function(a) a["PO7"]),
    use.names = FALSE)
Oz_listR <- unlist(lapply(resultListR, function(a) a["Oz"]),
    use.names = FALSE)
PO8_listR <- unlist(lapply(resultListR, function(a) a["PO8"]),
    use.names = FALSE)
```

```r
currentcounter <- 1
counter <- 0
FzMean <- list()
C3Mean <- list()
CzMean <- list()
C4Mean <- list()
PzMean <- list()
PO7Mean <- list()
OzMean <- list()
PO8Mean <- list()

# Calculate the mean of Shannon entropy
# values for adaptive participants
for (i in 1:37) {
    for (A in 1:length(RawEEGComplexSystemsRReadyAdaptiveR[[i]])) {
        counter <- counter + 1
    }

    FzMean[i] <- mean(Fz_listA[currentcounter:(currentcounter +
        counter)])
    C3Mean[i] <- mean(C3_listA[currentcounter:(currentcounter +
        counter)])
    CzMean[i] <- mean(Cz_listA[currentcounter:(currentcounter +
        counter)])
    C4Mean[i] <- mean(C4_listA[currentcounter:(currentcounter +
        counter)])
    PzMean[i] <- mean(Pz_listA[currentcounter:(currentcounter +
        counter)])
    PO7Mean[i] <- mean(PO7_listA[currentcounter:(currentcounter +
        counter)])
    OzMean[i] <- mean(Oz_listA[currentcounter:(currentcounter +
        counter)])
    PO8Mean[i] <- mean(PO8_listA[currentcounter:(currentcounter +
        counter)])

    currentcounter <- currentcounter + counter
    counter <- 0
}

currentcounter <- 1
counter <- 0
FzMeanRandom <- list()
C3MeanRandom <- list()
CzMeanRandom <- list()
C4MeanRandom <- list()
PzMeanRandom <- list()
PO7MeanRandom <- list()
OzMeanRandom <- list()
PO8MeanRandom <- list()

# Calculate the mean of Shannon entropy
# values for random participants
for (i in 1:37) {
    counter <- 0

    for (R in 1:length(RawEEGComplexSystemsRReadyAdaptiveR[[i]])) {
        counter <- counter + 1
    }

    FzMeanRandom[i] <- mean(Fz_listR[currentcounter:(currentcounter +
        counter)])
    C3MeanRandom[i] <- mean(C3_listR[currentcounter:(currentcounter +
        counter)])
    CzMeanRandom[i] <- mean(Cz_listR[currentcounter:(currentcounter +
        counter)])
    C4MeanRandom[i] <- mean(C4_listR[currentcounter:(currentcounter +
        counter)])
    PzMeanRandom[i] <- mean(Pz_listR[currentcounter:(currentcounter +
        counter)])
    PO7MeanRandom[i] <- mean(PO7_listR[currentcounter:(currentcounter +
        counter)])
    OzMeanRandom[i] <- mean(Oz_listR[currentcounter:(currentcounter +
        counter)])
    PO8MeanRandom[i] <- mean(PO8_listR[currentcounter:(currentcounter +
        counter)])

    currentcounter <- currentcounter + counter
    counter <- 0
}

# Combine the mean values for adaptive
# and random participants
Fz <- c(FzMean, FzMeanRandom)
C3 <- c(C3Mean, C3MeanRandom)
Cz <- c(CzMean, CzMeanRandom)
C4 <- c(C4Mean, C4MeanRandom)
Pz <- c(PzMean, PzMeanRandom)
PO7 <- c(PO7Mean, PO7MeanRandom)
Oz <- c(OzMean, OzMeanRandom)
PO8 <- c(PO8Mean, PO8MeanRandom)

# Create a factor variable for the
# condition ('A' and 'R')
condition <- factor(rep(c("A", "R"), each = length(FzMean)))

# Combine all variables into a data
# frame
data <- data.frame(Fz = unlist(Fz), C3 = unlist(C3),
    Cz = unlist(Cz), C4 = unlist(C4), Pz = unlist(Pz),
    PO7 = unlist(PO7), Oz = unlist(Oz), PO8 = unlist(PO8),
    condition = condition)

# Reshape the data into a long format
melted_data <- melt(data, id.vars = "condition",
    variable.name = "Channel", value.name = "Value")
```

```
# Create the boxplot using ggplot2
p <- ggplot(melted_data, aes(x = condition,
    y = Value, fill = condition)) + geom_boxplot() +
    facet_wrap(~Channel, ncol = 4) + ylim(3,
    NA) + scale_fill_manual(values = c("lightblue",
    "red"), labels = c("A", "R")) + labs(title = "Shannon Entropy per channel between conditions",
    x = "Channel", y = "Shannon Entropy") +
    theme_bw()


# Add significance annotations with
# adjusted label position
p + stat_compare_means(aes(group = condition),
    size = 3.2, method = "wilcox", label.y = 3)
```



These results suggest that there are no significant differences within the channels between the conditions (Due to Bonferonni correction, the significance threshold is 0.00625).

Sample Entropy

Sample entropy is a measure of the complexity or irregularity present in a signal (Delgado-Bonal, A. & Marshak, A., 2019). It is a measure of the complexity or irregularity specifically designed for time series data. It focuses on capturing the patterns and predictability within a sequence of data points. Sample entropy considers the likelihood that similar patterns of data points will be followed by other similar patterns within a specified tolerance or distance.

Sample Entropy is calculated using the following steps:

1. **Determine the length of the pattern:** Decide on the length of the patterns you want to compare in the time series. This is denoted by "edim," which stands for embedding dimension. It represents the number of consecutive data points in a pattern that will be compared.
2. **Set a tolerance level:** Choose a tolerance level or threshold, denoted by "r," which determines the maximum acceptable difference between two patterns for them to be considered similar. This value is often determined based on the characteristics of the time series data and the desired level of sensitivity. When you increase r, your entropy will most likely go down, as more patterns are deemed similar.
3. **Create vectors of length "edim" from the time series:** Slide a window of length "edim" across the time series data and extract subsequences of length "edim."
4. **Define the distance between vectors:** Calculate the distance or similarity between two vectors using a suitable metric such as Euclidean distance. The distance between vectors x and y is denoted as "d(x, y)."
5. **Compute the number of similar vector pairs:** For each vector, count the number of other vectors within the tolerance level "r" that are similar to it. Two vectors are considered similar if the distance between them is less than or equal to "r". Remember this as x.
6. **Compute the number of similar vector pairs with an additional vector:** Repeat the previous step, but this time include an additional vector. This value represents the number of vector pairs that remain similar when an additional data point is included in the pattern.Remember this as y.
7. **Calculate the probability of similarity:** Compute the probability of finding similar vector pairs by: taking the values from 6 and dividing this by the sum of the values of P = (y / (y + x))
8. **Repeat steps 5-7 for patterns of increasing length:** Repeat steps 5-7 for patterns of length "edim+1" to calculate the probability of similarity at the next scale. This step helps to capture the complexity of the time series at different scales.
9. **Repeat steps 5-8 for different pattern lengths:** Iterate over different values of "edim" to create a series of probabilities for various pattern lengths. This allows for a comprehensive analysis of the time series complexity at multiple scales.
10. **Calculate the sample entropy**: Finally, calculate the sample entropy by taking the negative natural logarithm of the ratio of the probabilities:

    Sample Entropy = -log(P)

Sample Entropy over all participants

Running this code cell takes several hours to run. It was ran before knitting, meaning that the results were stored, but not printed in the final knitted version.

```r
channellist <- c("Fz", "C3", "Cz", "C4",
    "Pz", "PO7", "Oz", "PO8")

## SampleEntropy
SampleEntropyAdaptiveChannels <- vector(mode = "list",
    length = length(channellist))

SampleEntropyAdaptiveParticipants <- vector(mode = "list",
    length = 37)
```

```r
SampleEntropyRandomChannels <- vector(mode = "list",
    length = length(channellist))

SampleEntropyRandomParticipants <- vector(mode = "list",
    length = 37)
# Loop over subjects
for (i in 1:37) {
    print(i)
    SampleEntropyAdaptiveEvents <- vector(mode = "list",
        length = length(RawEEGComplexSystemsRReadyAdaptiveR[[i]]))

    # Loop over trials
    for (A in 1:length(RawEEGComplexSystemsRReadyAdaptiveR[[i]])) {

        print(A)
        for (channel in channellist) {
            print(channel)
            AdaptiveSignal <- RawEEGComplexSystemsRReadyAdaptiveR[[i]][[A]][[channel]]

            e1A <- sample_entropy(AdaptiveSignal,
                edim = 2, r = 0.2 * sd(AdaptiveSignal),
                tau = 1)
            SampleEntropyAdaptiveChannels[[channel]] <- append(SampleEntropyAdaptiveChannels[[channel]]
                e1A)
        }
        SampleEntropyAdaptiveEvents[[A]] <- append(SampleEntropyAdaptiveEvents[[A]],
            SampleEntropyAdaptiveChannels)
        SampleEntropyAdaptiveChannels <- vector(mode = "list",
            length = length(channellist))
    }

    SampleEntropyRandomEvents <- vector(mode = "list",
        length = length(RawEEGComplexSystemsRReadyRandomR[[i]]))
    for (R in 1:length(RawEEGComplexSystemsRReadyRandomR[[i]])) {
        print(R)
        for (channel in channellist) {
            print(channel)
            RandomSignal <- RawEEGComplexSystemsRReadyRandomR[[i]][[R]][[channel]]

            e1R <- sample_entropy(RandomSignal,
                edim = 2, r = 0.2 * sd(RandomSignal),
                tau = 1)
            SampleEntropyRandomChannels[[channel]] <- append(SampleEntropyRandomChannels[[channel]],
                e1R)
        }
        SampleEntropyRandomEvents[[R]] <- append(SampleEntropyRandomEvents[[R]],
            SampleEntropyRandomChannels)
        SampleEntropyRandomChannels <- vector(mode = "list",
            length = length(channellist))
    }
    SampleEntropyAdaptiveParticipants[[i]] <- append(SampleEntropyAdaptiveParticipants[[i]],
        SampleEntropyAdaptiveEvents)
    SampleEntropyRandomParticipants[[i]] <- append(SampleEntropyRandomParticipants[[i]],
        SampleEntropyRandomEvents)
}



length(Fz_listA)
length(Fz_listR)

length(Fz_listShannonEntr)
Fz_listShannonEntr


Fz_listA <- unlist(lapply(resultListA, function(a) a["Fz"]),
    use.names = FALSE)
C3_listA <- unlist(lapply(resultListA, function(a) a["C3"]),
    use.names = FALSE)
Cz_listA <- unlist(lapply(resultListA, function(a) a["Cz"]),
    use.names = FALSE)
C4_listA <- unlist(lapply(resultListA, function(a) a["C4"]),
    use.names = FALSE)
Pz_listA <- unlist(lapply(resultListA, function(a) a["Pz"]),
    use.names = FALSE)
PO7_listA <- unlist(lapply(resultListA, function(a) a["PO7"]),
    use.names = FALSE)
Oz_listA <- unlist(lapply(resultListA, function(a) a["Oz"]),
    use.names = FALSE)
PO8_listA <- unlist(lapply(resultListA, function(a) a["PO8"]),
    use.names = FALSE)

Fz_listR <- unlist(lapply(resultListR, function(a) a["Fz"]),
    use.names = FALSE)
C3_listR <- unlist(lapply(resultListR, function(a) a["C3"]),
    use.names = FALSE)
Cz_listR <- unlist(lapply(resultListR, function(a) a["Cz"]),
    use.names = FALSE)
C4_listR <- unlist(lapply(resultListR, function(a) a["C4"]),
    use.names = FALSE)
Pz_listR <- unlist(lapply(resultListR, function(a) a["Pz"]),
    use.names = FALSE)
PO7_listR <- unlist(lapply(resultListR, function(a) a["PO7"]),
    use.names = FALSE)
Oz_listR <- unlist(lapply(resultListR, function(a) a["Oz"]),
    use.names = FALSE)
PO8_listR <- unlist(lapply(resultListR, function(a) a["PO8"]),
    use.names = FALSE)

Fz_listSampleEntropy <- c(Fz_listA, Fz_listR)
C3_listSampleEntropy <- c(C3_listA, C3_listR)
Cz_listSampleEntropy <- c(Cz_listA, Cz_listR)
C4_listSampleEntropy <- c(C4_listA, C4_listR)
Pz_listSampleEntropy <- c(Pz_listA, Pz_listR)
PO7_listSampleEntropy <- c(PO7_listA, PO7_listR)
Oz_listSampleEntropy <- c(Oz_listA, Oz_listR)
```

```
PO8_listSampleEntropy <- c(PO8_listA, PO8_listR)


write.csv(Fz_listSampleEntropy, file = "Fz_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(C3_listSampleEntropy, file = "C3_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(Cz_listSampleEntropy, file = "Cz_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(C4_listSampleEntropy, file = "C4_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(Pz_listSampleEntropy, file = "Pz_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(PO7_listSampleEntropy, file = "PO7_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(Oz_listSampleEntropy, file = "Oz_listShannonEntr2.csv",
    row.names = FALSE)
write.csv(PO8_listSampleEntropy, file = "PO8_listShannonEntr2.csv",
    row.names = FALSE)
```

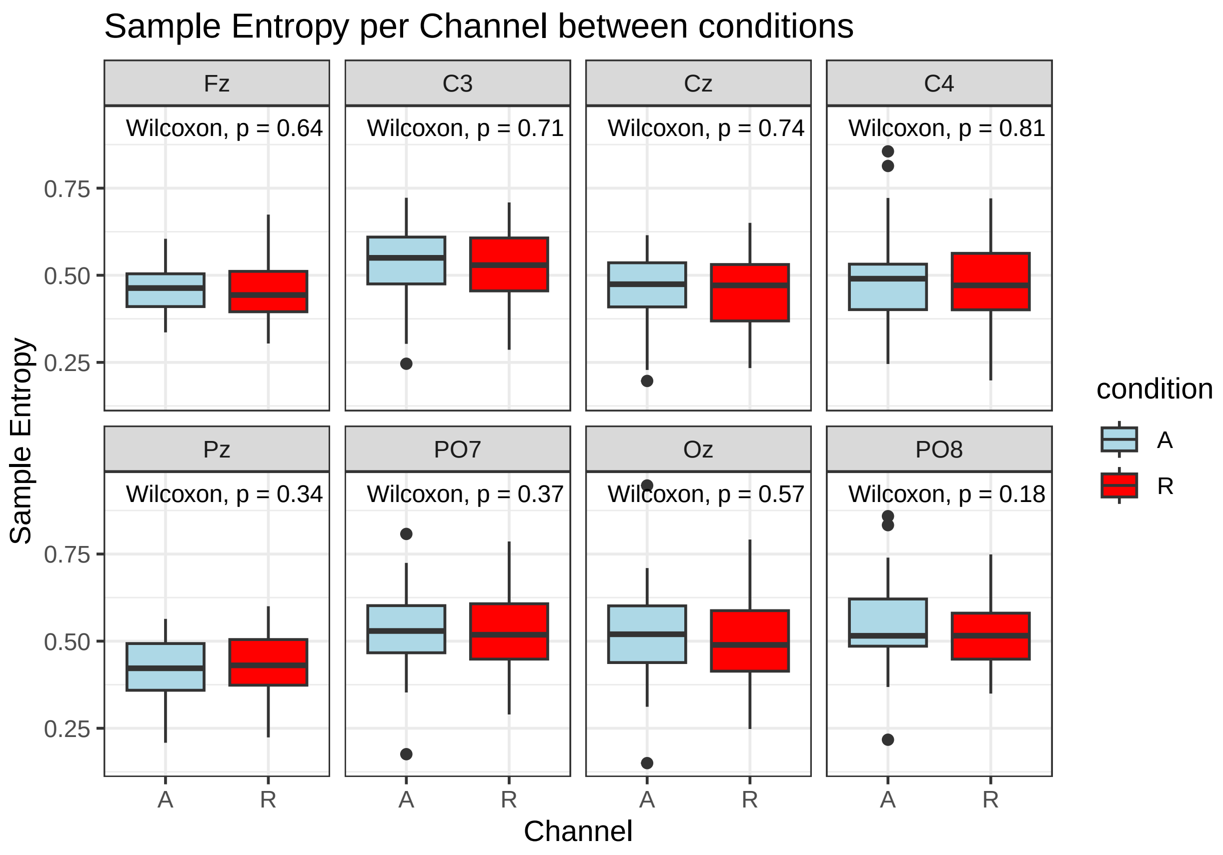
We load the data directly here:

```
Fz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Fz_listSampleEntr.csv")
C3_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/C3_listSampleEntr.csv")
Cz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Cz_listSampleEntr.csv")
C4_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/C4_listSampleEntr.csv")
Pz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Pz_listSampleEntr.csv")
PO7_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/PO7_listSampleEntr.csv")
Oz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Oz_listSampleEntr.csv")
PO8_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/PO8_listSampleEntr.csv")
```

We have averaged the Sample entropy over the conditions per participant so that we have an average Sample entropy per participant. The code was hidden, as it has no added value, and only clutters the notebook. The averaged data will be used for the following part of the notebook.

Boxplot of channels of Sample Entropy between conditions

```
# Create a factor variable for the
# condition ('A' and 'R')
condition <- factor(rep(c("A", "R"), each = length(FzMeanSampleEntr)/2))

# Combine all variables into a data
# frame
data <- data.frame(Fz = unlist(FzMeanSampleEntr),
    C3 = unlist(C3MeanSampleEntr), Cz = unlist(CzMeanSampleEntr),
    C4 = unlist(C4MeanSampleEntr), Pz = unlist(PzMeanSampleEntr),
    PO7 = unlist(PO7MeanSampleEntr), Oz = unlist(OzMeanSampleEntr),
    PO8 = unlist(PO8MeanSampleEntr), condition = condition)

# Reshape the data into a long format
melted_data <- reshape2::melt(data, id.vars = "condition",
    variable.name = "Channel", value.name = "Value")

# Create the boxplot using ggplot2
p <- ggplot2::ggplot(melted_data, aes(x = condition,
    y = Value, fill = condition)) + ggplot2::geom_boxplot() +
    ggplot2::facet_wrap(~Channel, ncol = 4) +
    ggplot2::scale_fill_manual(values = c("lightblue",
        "red"), labels = c("A", "R")) + ggplot2::labs(title = "Sample Entropy per Channel between condi
    x = "Channel", y = "Sample Entropy") +
    ggplot2::theme_bw()

p + stat_compare_means(aes(group = condition),
    size = 3.2, method = "wilcox", label.y = 0.9,
    paired = TRUE)
```

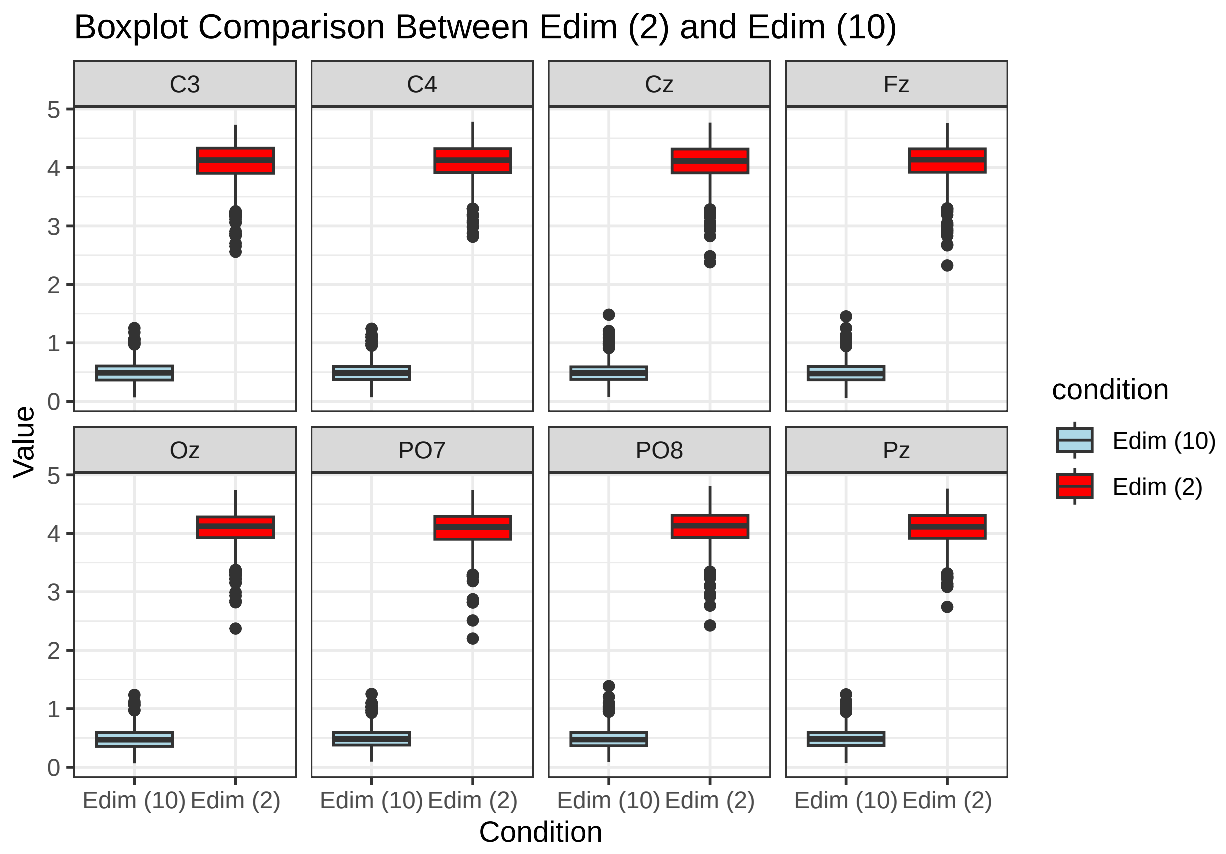


As seen in the plot above, it does not seem that the entropy in the signals is significantly different between conditions. This means that the insignificant difference in sample entropy shows similar patterns or complexity within the EEG signal between conditions.

```
Fz_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/Fz_listSampleEntr2.csv")
C3_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/C3_listSampleEntr2.csv")
Cz_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/Cz_listSampleEntr2.csv")
C4_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/C4_listSampleEntr2.csv")
Pz_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/Pz_listSampleEntr2.csv")
PO7_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/PO7_listSampleEntr2.csv")
Oz_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/Oz_listSampleEntr2.csv")
```

```r
PO8_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New folder/PO8_listSampleEntr2.csv")

Fz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Fz_listSampleEntr.csv")
C3_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/C3_listSampleEntr.csv")
Cz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Cz_listSampleEntr.csv")
C4_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/C4_listSampleEntr.csv")
Pz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Pz_listSampleEntr.csv")
PO7_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/PO7_listSampleEntr.csv")
Oz_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/Oz_listSampleEntr.csv")
PO8_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New folder/PO8_listSampleEntr.csv")


# Create a list of variables
variables2 <- c(Fz_listSampleEntr$x, C3_listSampleEntr$x,
    Cz_listSampleEntr$x, C4_listSampleEntr$x,
    Pz_listSampleEntr$x, PO7_listSampleEntr$x,
    Oz_listSampleEntr$x, PO8_listSampleEntr$x)

variables1 <- c(Fz_listSampleEntr2$x, C3_listSampleEntr2$x,
    Cz_listSampleEntr2$x, C4_listSampleEntr2$x,
    Pz_listSampleEntr2$x, PO7_listSampleEntr2$x,
    Oz_listSampleEntr2$x, PO8_listSampleEntr2$x)


# Combine the two lists of variables
allVariables <- c(variables1, variables2)

# Create a factor variable indicating
# the source (variables1 or variables2)
condition <- factor(rep(c("Edim (2)", "Edim (10)"),
    each = length(variables1)))

# Combine all variables into a data
# frame
data <- data.frame(Channel = rep(c("Fz",
    "C3", "Cz", "C4", "Pz", "PO7", "Oz",
    "PO8"), times = 2), Value = unlist(allVariables),
    condition = condition)

# Create the boxplot using ggplot2
p <- ggplot(data, aes(x = condition, y = Value,
    fill = condition)) + geom_boxplot() +
    facet_wrap(~Channel, ncol = 4) + scale_fill_manual(values = c("lightblue",
    "red"), labels = c("Edim (10)", "Edim (2)")) +
    labs(title = "Boxplot Comparison Between Edim (2) and Edim (10)",
        x = "Condition", y = "Value") + theme_bw()

# Add statistical comparisons using
# stat_compare_means() p +
# stat_compare_means(aes(group =
# condition), size = 3.2, method =
# 'wilcox', label.y = 0.90, paired =
# TRUE)
```

```r
p
```



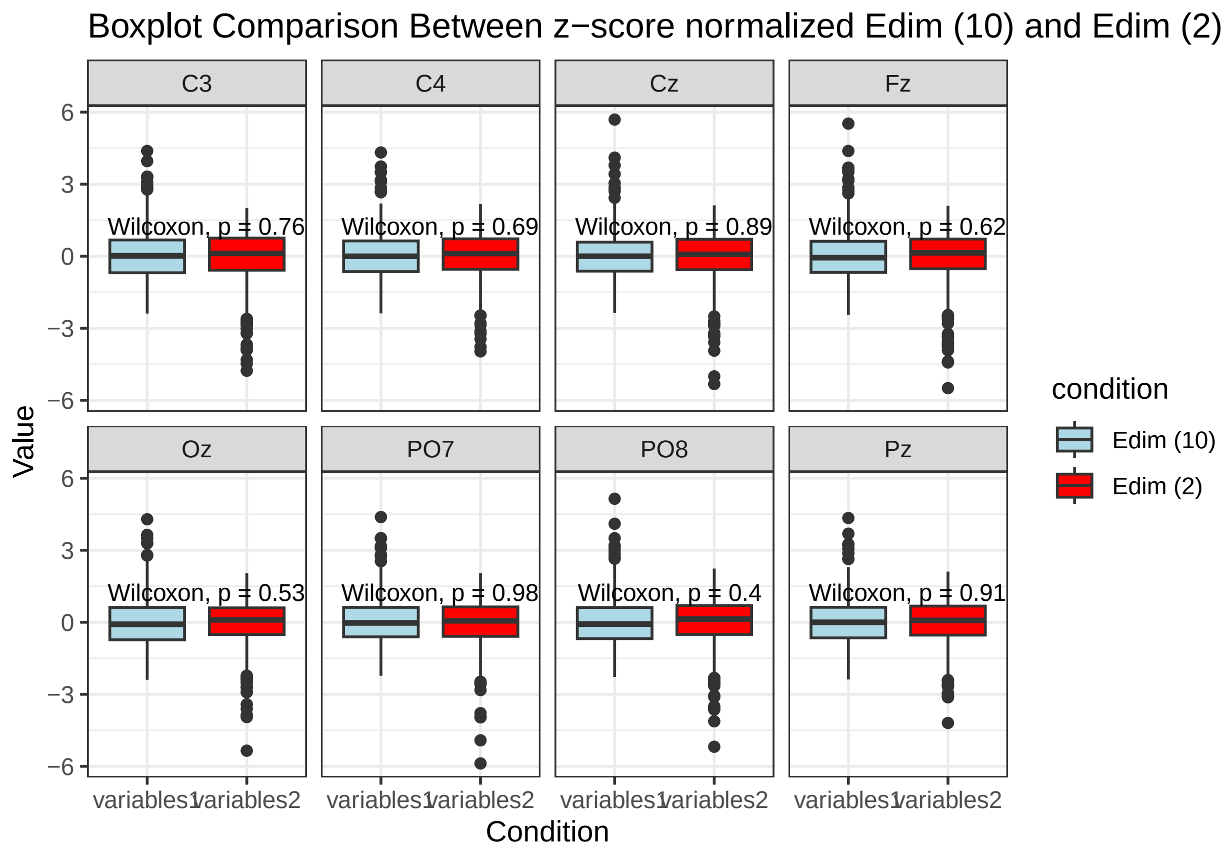
When we increase the embedding dimension (edim), it has an intriguing impact on entropy as it tends to make the entropy values lower. As a result, the overall uncertainty or randomness decreases.

The decrease in entropy with a higher edim is likely due to a couple of factors. First, the longer patterns have a smoothing effect. By considering more data points in each pattern, the smaller fluctuations and noise tend to average out, making the patterns appear more regular and less chaotic. This reduction in randomness is likely reflected in the lower entropy values.

Moreover, longer patterns might overlook the finer details and nuances present in shorter patterns. It's like looking at a broader overview rather than focusing on the intricate aspects. These smaller variations contribute to the overall complexity and irregularity of the time series. When we extend the patterns, we might miss capturing these small irregularities, resulting in lower entropy values.

Based on previous literature, an Edim of 10 is too high for such a short time series (Srinivasan, S., & Rajagopalan, V., 2018). We can clearly see that the entropy is drastically affected by this measure, highlighting the importance of having a proper Edim value.

```r
variables1 <- c(Fz_listSampleEntr$x, C3_listSampleEntr$x,
    Cz_listSampleEntr$x, C4_listSampleEntr$x,
    Pz_listSampleEntr$x, PO7_listSampleEntr$x,
    Oz_listSampleEntr$x, PO8_listSampleEntr$x)

variables2 <- c(Fz_listSampleEntr2$x, C3_listSampleEntr2$x,
    Cz_listSampleEntr2$x, C4_listSampleEntr2$x,
    Pz_listSampleEntr2$x, PO7_listSampleEntr2$x,
    Oz_listSampleEntr2$x, PO8_listSampleEntr2$x)

variables1 <- scale(variables1)
variables2 <- scale(variables2)
# Combine the two lists of variables
allVariables <- c(variables1, variables2)

# Create a factor variable indicating
# the source (variables1 or variables2)
condition <- factor(rep(c("variables1", "variables2"),
    each = length(variables1)))

# Combine all variables into a data
# frame
data <- data.frame(Channel = rep(c("Fz",
    "C3", "Cz", "C4", "Pz", "PO7", "Oz",
    "PO8"), times = 2), Value = unlist(allVariables),
    condition = condition)

# Create the boxplot using ggplot2
p <- ggplot(data, aes(x = condition, y = Value,
    fill = condition)) + geom_boxplot() +
    facet_wrap(~Channel, ncol = 4) + scale_fill_manual(values = c("lightblue",
    "red"), labels = c("Edim (10)", "Edim (2)")) +
    labs(title = "Boxplot Comparison Between z-score normalized Edim (10) and Edim (2)",
        x = "Condition", y = "Value") + theme_bw()

# Add statistical comparisons using
# stat_compare_means()
p + stat_compare_means(aes(group = condition),
    size = 3.2, method = "wilcox", label.y = 0.9,
    paired = TRUE)
```



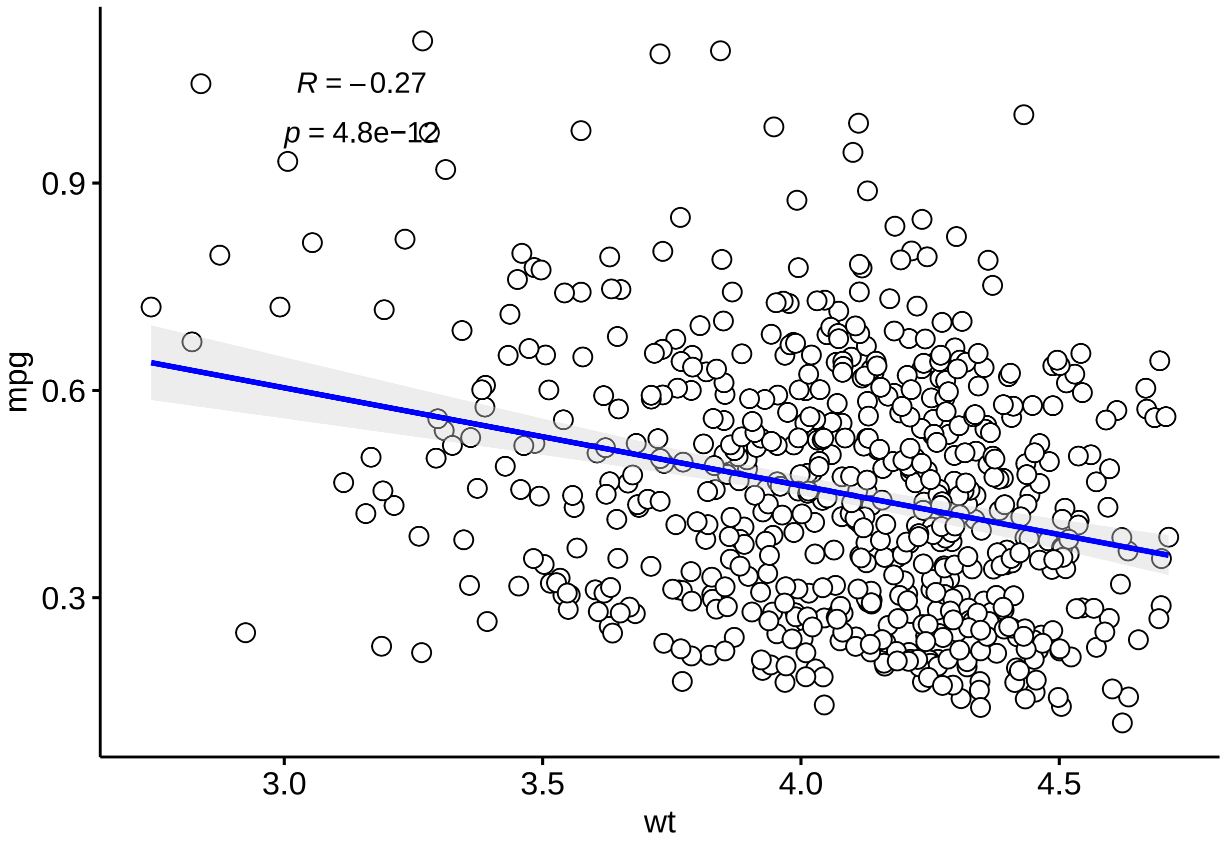
What is interesting however is that when we scale the found entropy based on z-score normalization, we get values that are not significantly different from Edim 2. Do note that the p values are not Nan or 1, meaning that the values are somewhat different. This could mean that the entropy in participants "function" similarly when comparing within the same scale. While the entropy is significantly different in terms of magnitude based on different scales, it becomes insignificantly different when we remove the magnitude. In other words, this indicates a consistent pattern of entropy dynamics or trends within the same scale, regardless of the individual differences in absolute magnitude. This is in line with what was found by Delgado-Bonal and Marshak (2019).

Correlation between Sample Entropy and MFDFA Distance

As explained in the module about Scales and Fractals, the Distance of the MFDFA Spectrum shows us the degree of multifractality on different scales. Given our previous finding, suggesting that perhaps Entropy in EEG changes in a similar way over participants, it could be insightful to 1) see whether a correlation exists between Sample Entropy and MFDFA, and 2) see whether this correlation stays the same over different Edims of Entropy.

```
library(ggpubr)

# Read the data for entropy and
# multifractal distance
C3_listSampleEntr2 <- read.csv("C:/Users/caty-/Downloads/New Folder/C3_listSampleEntr2.csv")
C3_listMFDFA <- read.csv("C:/Users/caty-/Downloads/New folder/C3_listMFDFA.csv")
C3_listSampleEntr2 <- C3_listSampleEntr2$x
C3_listMFDFA <- C3_listMFDFA$x
```

```
# Combine the data into a data frame
df <- data.frame(wt = C3_listSampleEntr2,
    mpg = C3_listMFDFA)

# Create the scatter plot with
# regression line and correlation
# coefficient
ggscatter(df, x = "wt", y = "mpg", color = "black",
    shape = 21, size = 3, add = "reg.line",
    add.params = list(color = "blue", fill = "lightgray"),
    conf.int = TRUE, cor.coef = TRUE, cor.coeff.args = list(method = "pearson",
        label.x = 3, label.sep = "\n"))
```

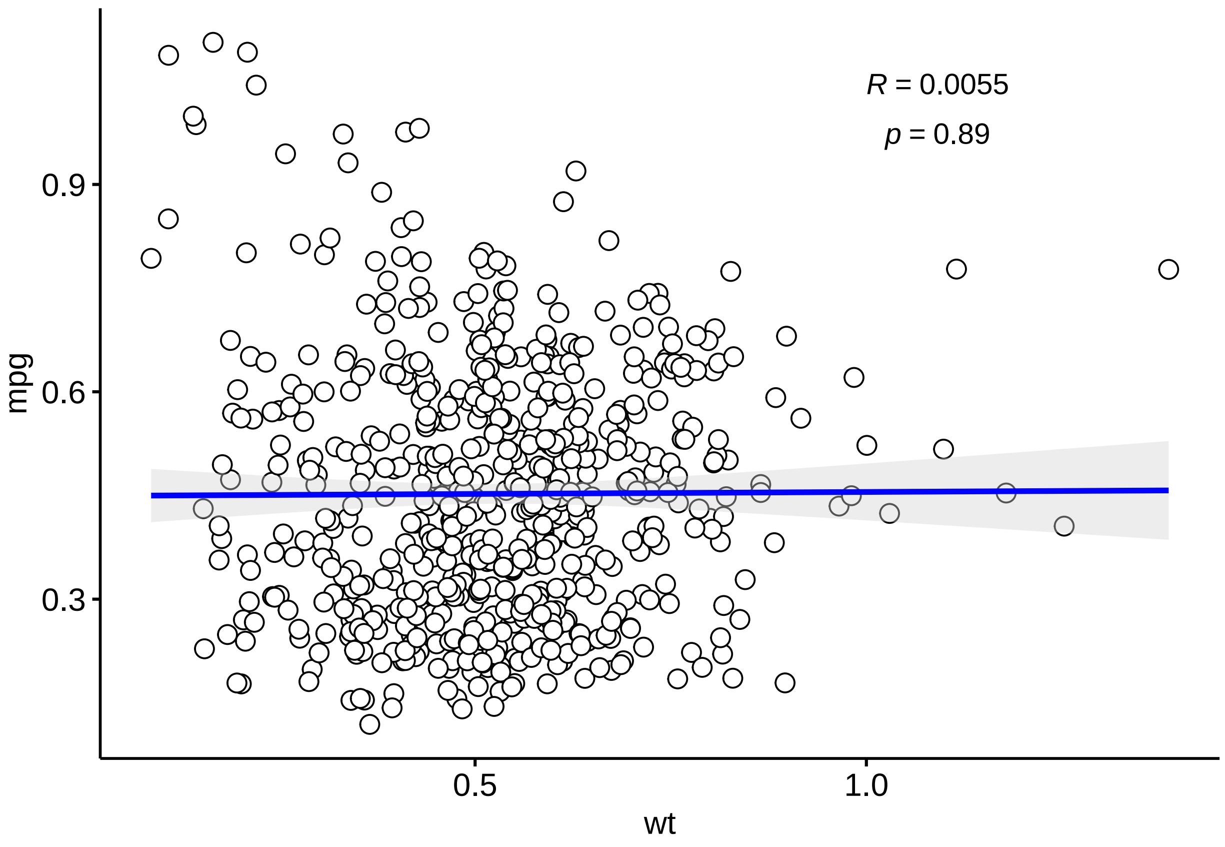


The correlation between MFDFA and Entropy suggests that changes in the complexity or irregularity of the time series, as captured by the Entropy measure, are associated with changes in the scaling properties and long-range correlations detected by MFDFA. This implies that alterations in the unpredictability or randomness of patterns are related to the statistical properties and self-similarity of the time series.

We now take a look at the correlation plot of the Sample Entropy with Edim10 and the Distance of the MFDFA Spectrum to see whether the same results can be found.

```
# Read the data for entropy and
# multifractal distance
C3_listSampleEntr <- read.csv("C:/Users/caty-/Downloads/New Folder/C3_listSampleEntr.csv")
C3_listMFDFA <- read.csv("C:/Users/caty-/Downloads/New Folder/C3_listMFDFA.csv")
C3_listSampleEntr <- C3_listSampleEntr$x
```

```
C3_listMFDFA <- C3_listMFDFA$x

# Combine the data into a data frame
df <- data.frame(wt = C3_listSampleEntr,
    mpg = C3_listMFDFA)

# Create the scatter plot with
# regression line and correlation
# coefficient
ggscatter(df, x = "wt", y = "mpg", color = "black",
    shape = 21, size = 3, add = "reg.line",
    add.params = list(color = "blue", fill = "lightgray"),
    conf.int = TRUE, cor.coef = TRUE, cor.coeff.args = list(method = "pearson",
        label.x = 1, label.sep = "\n"))
```



As we can see in the plot it is not the case that Sample Entropy with Edim10 correlates with the Distance in MFDFA.

Firstly, the fact that Edim2 exhibits a significant negative correlation with MFDFA implies that changes in complexity and irregularity captured by Sample Entropy, when calculated with an Edim of 2, are associated with alterations in the scaling properties and long-range correlations detected by MFDFA. This suggests that as the entropy of the time series increases or decreases, there are corresponding trends in the scaling behavior and long-range correlations. This observation indicates a relationship between sample entropy dynamics and the statistical properties of the data at smaller scales.

On the other hand, the lack of significance in the correlation between Edim10 and MFDFA unveils an intriguing aspect. It suggests that the relationship between entropy calculated with an Edim of 10 and the scaling properties and long-range correlations detected by MFDFA is less pronounced. This implies that changes in complexity and irregularity captured by Sample Entropy with Edim10 may not have a strong influence on the scaling behavior and long-range correlations of the time series. This is in line with our hypothesis that an increase in the Edim has a smoothing effect and overlooks finer details.

Now, let's tie this to the distance of the MFDFA plot. The MFDFA plot represents the relationship between the segment size and the corresponding fluctuation of the time series. It provides insights into the scaling properties and long-range correlations exhibited by the data. The fact that the correlation between Edim2 and MFDFA is significant while the correlation between Edim10 and MFDFA is not, suggests that the choice of embedding dimension can impact the proximity of the entropy measure to the scaling behavior and long-range correlations represented by the MFDFA plot.

This observation implies that the embedding dimension used in calculating entropy plays a critical role in capturing the relevant dynamics of the time series. Different embedding dimensions emphasize different aspects of the data and may bring entropy measures closer or farther from the scaling behavior detected by MFDFA. The choice of embedding dimension can be seen as a magnifying glass, zooming in on specific characteristics and highlighting their relationship with the scaling properties and long-range correlations.

Further analysis should look at different Edim's and their relation to the MFDFA Spectrum Distance, to see whether these results and conclusions hold. Furthermore, different q values of MFDFA should be examined. It could be the case that with the limited number of q values we tried during the Fractal module, the long-range scaling properties and long-range correlation are not captured, and thus don't correlate with the "Large-scale" Edim of Sample Entropy values.

References

Antonio, Narzo FD (2019). *tseriesChaos: Analysis of Nonlinear Time Series.* R package version 0.1-13.1, https://CRAN.R-project.org/package=tseriesChaos.

Delgado-Bonal, A., & Marshak, A. (2019). Approximate entropy and sample entropy: A comprehensive tutorial. Entropy, 21(6), 541.

Garcia C (2022). *nonlinearTseries: Nonlinear Time Series Analysis.* R package version 0.2.12, https://CRAN.R-project.org/package=nonlinearTseries.

Hadley Wickham (2007). Reshaping Data with the reshape Package. Journal of Statistical Software, 21(12), 1-20. URL http://www.jstatsoft.org/v21/i12/.

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

Kassambara A (2023). *ggpubr: 'ggplot2' Based Publication Ready Plots.* R package version 0.6.0, https://CRAN.R-project.org/package=ggpubr.

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation. arXiv preprint arXiv:2001.08025. https://arxiv.org/pdf/2001.08025.pdf

Saraiva, P. (2023). On Shannon entropy and its applications. Kuwait Journal of Science, in press. https://doi.org/10.1016/j.kjs.2023.05.004

Shannon, C. E. (1948). A mathematical theory of communication. Bell System Technical Journal, 27(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Srinivasan, S., & Rajagopalan, V. (2018). Efficient Methods for Calculating Sample Entropy in Time Series Data. Procedia Computer Science, 143, 103–110. https://doi.org/10.1016/j.procs.2018.10.476

Ushey K, Allaire J, Tang Y (2023). *reticulate: Interface to 'Python'.* R package version 1.28, https://CRAN.R-project.org/package=reticulate.

Wickham H, Henry L (2023). *purrr: Functional Programming Tools.* R package version 1.0.1, https://CRAN.R-project.org/package=purrr.